Estandarización de código para el desarrollo del sistema Inbo.

|  |  |
| --- | --- |
|  |  |

Universidad Veracruzana (UV)

De La Cruz Díaz Adolfo Ángel

Josafat Murillo Hernández

Contenido

[1.-Introducción. 4](#_Toc507594765)

[2.-Organización de paquetes. 4](#_Toc507594766)

[2.1-Archivos fuente 4](#_Toc507594767)

[2.1.1-Comentarios de encabezado. 4](#_Toc507594768)

[2.1.2-Sentencias package e import 5](#_Toc507594769)

[3.-Declaraciones de clases e interfaces. 5](#_Toc507594770)

[4.-Identacion 6](#_Toc507594771)

[4.1-Longitud de las líneas. 6](#_Toc507594772)

[4.1.1-Ruptura de líneas. 6](#_Toc507594773)

[5.-Declaraciones 7](#_Toc507594774)

[5.1.-Declaraciones por línea 7](#_Toc507594775)

[5.3.-Colocación 7](#_Toc507594776)

[5.4.-Declaracion de clases e interfaces 8](#_Toc507594777)

[6. Sentencias 8](#_Toc507594778)

[6.1 Sentencias simples 8](#_Toc507594779)

[6.2 Sentencias compuestas 9](#_Toc507594780)

[6.3 Sentencias de retorno 9](#_Toc507594781)

[6.4 Sentencias if, if-else, if else-if else 9](#_Toc507594782)

[6.5 Sentencias for 9](#_Toc507594783)

[6.6 Sentencias while 9](#_Toc507594784)

[6.7 Sentencias do-while 10](#_Toc507594785)

[6.8 Sentencias switch 10](#_Toc507594786)

[6.9 Sentencias try-catch 10](#_Toc507594787)

[7.-Espacios en blanco 10](#_Toc507594788)

[7.1.-Lineas en blanco 10](#_Toc507594789)

[7.2.-Espacios en blanco 11](#_Toc507594790)

[8.- Convenciones de nombres 11](#_Toc507594791)

[8.1 Clases 11](#_Toc507594792)

[8.2 Interfaces 12](#_Toc507594793)

[8.3 Métodos 12](#_Toc507594794)

[8.4 Variables 12](#_Toc507594795)

[8.5 Constantes 12](#_Toc507594796)

[9 - Hábitos de programación 12](#_Toc507594797)

[9.1 Proporcionando acceso a variables de instancia y de clase 12](#_Toc507594798)

[9.2 Referencias a variables y métodos de clase 12](#_Toc507594799)

## **1.-Introducción.**

Este documento contiene una serie de instrucciones con las que debe ser realizado el proyecto del Sistema Automatizado de Academias. Es una estandarización de construcción de código, que hace que la codificación tenga robustez, consistencia, claridad, completo y simple. El documento está dividido en secciones que dan una explicación a fondo de cómo debe estar estructurado el código. Comenzando con la organización de paquetes, los archivos de código fuente, los tipos de identacion, como declarar las sentencias, los espacios en blanco, las convenciones de nombres para métodos, clases, interfaces y variables. El conjunto de todas estas ayudará al usuario encargado del mantenimiento al código brindándole facilidad para entender su estructura, tendrá más facilidad de lectura y podrá encontrar los errores o faltas de optimización más rápido.

## **2.-Organización de paquetes.**

Un archivo de código consiste de secciones que deben estar separadas por líneas en blanco y comentarios opcionales que identifican cada sección.

Los archivos de más de 2000 líneas son incómodos y deben ser evitados a toda costa.

**2.1-Archivos fuente.**

Cada archivo de código fuente .java contiene una única clase o interface con su debido encapsulamiento. Cuando algunas clases o interfaces privadas están asociadas a una clase pública, pueden ponerse en el mismo archivo que la clase pública. La clase o interfaz pública debe ser la primera clase o interface del archivo.

## **2.1.1-Comentarios de encabezado.**

Todos los archivos fuente deben comenzar con un comentario en el que se lista el nombre de la clase, nombre del programador información de la versión, fecha, y compañía:

/\*

\* Clase Principal

\*

\* Murillo Hernández Josafat; De La Cruz Díaz Adolfo Ángel

\*

\* Versión 0.1

\*

\* 27/02/2018

\*

\* Inbo

\*/

**Comentarios dentro del cuerpo del código**

Debe tener una estructura simple al con //, a diferencia del encabezado, debe ser un comentario claro y preciso, no importa la longitud de este, siempre y cuando sea relevante para conocer porque se hizo el código, no como funciona.

//Este bloque de código se creó por convicción de la muestra de un buen comentario.

For (int i = 0; ar; i++){

System.out.print(ar[i]);

}

**2.1.2-Sentencias package e import.**

La primera línea no-comentario de los archivos de código fuente Java es la sentencia package. Después de esta, pueden seguir varias sentencias import. Por ejemplo:

package java.awt;

import java.awt.peer.CanvasPeer;

**Nota**: El primer componente del nombre de un paquete único se escribe siempre en minúsculas.

## **3.-Declaraciones de clases e interfaces.**

|  |  |  |
| --- | --- | --- |
|  | **Declaración de clase o interface.** | **Notas.** |
| **1** | Comentario de documentación de la clase o interface (  /\*\* \* La clase Ejemplo ofrece … \*/  public class Ejemplo { …) | Cada clase deberá contener una breve descripción de lo que esta realiza en concreto, en caso de tener una clase privada en el mismo archivo, deberá contener la descripción de lo que realizan juntas y el por qué es necesario tenerlas en el mismo archivo. |
| **2** | Sentencia class o interface | Cada sentencia debe llevar su respectivo identificador, ya sea class o interface. |
| **3** | Variables de clase (static) | Primero las variables de clase public, después las protected, después las de nivel de paquete (sin modificador de acceso) , y despúes las private. |
| **4** | Constructores | Se deben declarar de forma explícita. |
| **5** | Métodos. | Estos métodos se deben agrupar por bloques de funcionalidad más que por visión o accesibilidad. Por ejemplo, un método de clase privado puede estar entre dos métodos públicos de instancia.  El objetivo es hacer el código más legible y comprensible. |

**Tabla 1.** Describe las partes de la declaración de una clase o interface, en el orden en que deberían aparecer.

**4.-Identacion.**

Se deben emplear cuatro espacios como unidad de indentación, esto es una fórmula de acuerdo al nivel en la jerarquía de condiciones o ciclos que es 4 \* N (N: Nivel de jerarquización).

## **4.1-Longitud de las líneas.**

Evitar las líneas de más de 7 parámetros cada una, ya que no son manejadas bien por muchas terminales y herramientas.

**Nota**: Ejemplos para uso en la documentación deben tener una longitud inferior, generalmente no más de 70 caracteres.

## **4.1.1-Ruptura de líneas.**

Cuando una expresión no entre en una línea, romperla de acuerdo con estos principios:

* Romper después de una coma.
* Romper antes de un operador.
* Alinear la nueva línea de código con el comienzo de la expresión al mismo nivel de la línea anterior.
* Si las reglas anteriores llevan a código confuso o a código que se aglutina en el margen derecho, indentar justo 8 espacios en su lugar.

Ejemplos de cómo romper la llamada a un método:

void sacarPromedio(int calificacionParcial1, int calificacionParcial2,

int calificacionParcial3);

int promedio = sacarPromedio(calificacionParcial1,calificacionParcial2,

calificacionParcial3);

## **5.-Declaraciones**

## **5.1.-Declaraciones por línea**

Se recomienda hacer una declaración por línea, ya que facilita los comentarios. Es decir, se tiene que hacer esto:

int nivelIdentación; // nivel de indentación

int tamañoTabla; // tamaño de la tabla

y no esto:

int nivel, tamaño;

**5.2.-Inicialización**

Inicializar las variables en donde se declaran, solo no aplica si esa variable depende su valor inicial de un cálculo que deba ocurrir, es decir:

int nivel = 0;

## **5.3.-Colocación**

Poner las declaraciones solo al inicio de un bloque de código (aquel que lleva de inicio “{” y fin “}”), la única excepción de esta regla serían los bucles for, donde estos pueden ser declarados dentro de ese bucle

void sumaNumeros (){

int numero = 0;

int suma = 0;

if (numero ¡= -1){

int numero2 = 0;

suma = numero + numero2;

}

}

## **5.4.-Declaracion de clases e interfaces**

Al codificar clases e interfaces de Java, se siguen las siguientes normas:

· Ningún espacio en blanco entre el nombre de un método y el paréntesis "(" que abre su lista de parámetros

· La llave de apertura "{" aparece al final de la misma línea de la sentencia declaración

· La llave de cierre "}" empieza una nueva línea indentada para ajustarse a su sentencia de apertura correspondiente, excepto cuando no existen sentencias entre ambas, que debe aparecer inmediatamente después de la de apertura "{"

class Ejemplo extends Object {

int ivar1;

int ivar2;

Ejemplo(int i, int j) {

ivar1 = i;

ivar2 = j;

}

int metodoVacio() {}

...

}

· Los métodos se separan con una línea en blanco

## **6. Sentencias**

## **6.1 Sentencias simples**

Cada línea debe contener como mucho una sentencia.

Ejemplo:

argv++; // Correcto

argc--; // Correcto

argv++; argc--; // EVITAR!

## **6.2 Sentencias compuestas**

Las sentencias compuestas son sentencias que contienen listas de sentencias encerradas entre llaves "{ sentencias }".

* Las sentencias encerradas deben indentarse un nivel más que la sentencia compuesta.
* La llave de apertura se debe poner al final de la linea que comienza la sentencia compuesta; la llave de cierre debe empezar una nueva línea y ser indentada al mismo nivel que el pincipio de la sentencia compuesta.
* Las llaves se usan en todas las sentencias, incluso las simples, cuando forman parte de una estructura de control, como en las sentencias if-else o for. Esto hace más sencillo añadir sentencias sin incluir bugs accidentalmente por olvidar las llaves.

## **6.3 Sentencias de retorno**

Una sentencia return con un valor no debe usar paréntesis a menos que hagan el valor de retorno más obvio de alguna manera. Ejemplo:

return;

return miDiscoDuro.size();

## **6.4 Sentencias if, if-else, if else-if else**

La clase de sentencias if-else no debe tener la siguiente estructura:

if (condicion) //EVITAR! ESTO OMITE LAS LLAVES {}!

## **6.5 Sentencias for**

Una sentencia for debe tener la siguiente forma:

for (inicializacion; condicion; actualizacion) {

sentencias;

}

Una sentencia for vacía (una en la que todo el trabajo se hace en las cláusulas de inicialización, condición, y actualización) debe tener la siguiente forma:

for (inicialización; condición; actualización);

Al usar el operador coma en la cláusula de inicialización o actualización de una sentencia for, evitar la complejidad de usar más de tres variables. Si se necesita, usar sentencias separadas antes de bucle for (para la cláusula de inicialización) o al final del bucle (para la cláusula de actualización).

## **6.6 Sentencias while**

Una sentencia while debe tener la siguiente forma:

while (condicion) {

sentencias;

}

Una sentencia while vacía debe tener la siguiente forma:

while (condicion);

## **6.7 Sentencias do-while**

Una sentencia do-while debe tener la siguiente forma:

do {

sentencias;

} while (condicion);

## **6.8 Sentencias switch**

Una sentencia switch debe tener la siguiente forma:

switch (condicion) {

case 1: sentencias; /\* este caso se propaga \*/

case 2: sentencias;

break;

case 3: sentencias;

break;

default: sentencias;

break;

}

Cada vez que un caso se propaga (no incluye la sentencia break), añadir un comentario donde la sentencia break se encontraría normalmente. Esto se muestra en el ejemplo anterior con el comentario /\* este caso se propaga \*/. Cada sentencia switch debe incluir un caso por defecto. El break en el caso por defecto es redundante, pero prevee que se propague por error si luego se añade otro caso.

## **6.9 Sentencias try-catch**

Una sentencia try-catch debe tener la siguiente forma:

try {

sentencias;

} catch (ExceptionClass e) {

sentencias;

}

Una sentencia try-catch puede ir seguida de un finally, cuya ejecución se ejecutará independientemente de que el bloque try se haya completado con éxito o no.

try {

sentencias;

} catch (ExceptionClass e) {

sentencias;

} finally {

sentencias;

}

## **7.-Espacios en blanco**

## **7.1.-Lineas en blanco**

Las líneas en blanco mejoran la facilidad de lectura separando secciones de código lógicamente relacionados. Se deben usar una línea en blanco en:

* Entre métodos
* Entre las variables locales de un método y su primera sentencia

## **7.2.-Espacios en blanco**

Se deben usar espacios en blanco en las siguientes circunstancias:

· Una palabra clave del lenguaje seguida por un paréntesis debe separarse por un espacio. Ejemplo:

while (true) {

...

}

Notar que no se debe usar un espacio en blanco entre el nombre de un método y su

paréntesis de apertura. Esto ayuda a distinguir palabras claves de llamadas a métodos.

· Debe aparecer un espacio en blanco después de cada coma en las listas de argumentos.

· Todos los operadores binarios excepto . se deben separar de sus operandos con espacios en blanco. Los espacios en blanco no deben separadar los operadores unarios, incremento ("++") y decremento ("--") de sus operandos. Ejemplo:

a += c + d;

a = (a + b) / (c \* d);

while (d++ == s++) {

n++;

}

printSize("el tamaño es " + foo + "\n");

· Las expresiones en una sentencia for se deben separar con espacios en blanco.

Ejemplo:

for (expr1; expr2; expr3)

## **8.- Convenciones de nombres**

## **8.1 Clases**

Los nombres de las clases deben ser sustantivos, cuando son compuestos tendrán la primera letra de cada palabra que lo forma en mayúsculas. Intentar mantener los nombres de las clases simples y descriptivas. Usar palabras completas, evitar acrónimos y abreviaturas (a no ser que la abreviatura sea mucho más conocida que el nombre completo, como URL o HTML). Ejemplo:

class Cliente;

class ImagenAnimada;

## **8.2 Interfaces**

Los nombres de las interfaces siguen la misma regla que las clases. Ejemplo:

interface ObjetoPersistente;

interface Almacen;

## **8.3 Métodos**

Los métodos deben ser verbos, cuando son compuestos tendrán la primera letra en minúscula, y la primera letra de la siguiente palabra será en mayúscula. Ejemplo:

ejecutar();

ejecutarRapido();

tomarFondo();

## **8.4 Variables**

Excepto las constantes, todas las instancias y variables de clase o método empezarán con minúscula. Las palabras internas que lo forman (si son compuestas) empiezan con su primera letra en mayúsculas. Los nombres de variables no deben empezar con los caracteres subguión "\_" o signo del dolar "$". Los nombres de las variables deben ser cortos pero con significado. La elección del nombre de una variable debe ser representativa, designado para indicar a un observador casual su función. Los nombres de variables de un solo caracter se deben evitar, excepto para variables índices temporales. Nombres comúnes para variables temporales son i, j, k, m, y n para enteros; c, d, y e para caracteres.

int i;

char c;

float miAnchura;

## **8.5 Constantes**

Los nombres de las variables declaradas como constantes deben ir totalmente en mayúsculas separando las palabras con un subguión ("\_").

static final int ANCHURA\_MINIMA = 4;

static final int ANCHURA\_MAXIMA = 999;

static final int COGER\_LA\_CPU = 1;

## **9 - Hábitos de programación**

## **9.1 Proporcionando acceso a variables de instancia y de clase**

No hacer nínguna variable de instancia o clase pública sin una buena razón. A menudo las variables de instancia no necesitan ser asignadas/consultadas explícitamente, a menudo esto sucede como efecto lateral de llamadas a métodos.

## **9.2 Referencias a variables y métodos de clase**

Evitar usar un objeto para acceder a una variable o método de clase (static). Usar el nombre de la clase en su lugar. Por ejemplo:

metodoDeClase(); //OK

UnaClase.metodoDeClase(); //OK

unObjeto.metodoDeClase(); //EVITAR!